04

Devolviendo el codigo 201

Transcripción

[00:00] Bien, ya vimos cómo actualizar correctamente siguiendo las buenas prácticas de Rest un objeto. Vamos con el siguiente que es el post. Vamos a guardar un nuevo recurso médico en nuestra base de datos.

[00:14] El post tiene una curiosidad y es que tiene que recordar un código específico, que es el 201, que es created. Entonces primero que todo voy a poner aquí, debo retornar 201 created, pero Post también debe retornar algo y esto es por la especificación que tiene Rest. Y esto no es solo para Spring, esto es para cualquier tipo de API, es algo universal, es un estándar.

[00:46] Rest una vez que crea el objeto, por ejemplo, si yo registro mi médico, yo debería retornar en un header la URL donde tú puedes encontrar este médico. Entonces donde encontrar al médico. ¿Por qué? Porque por buena práctica, si ya está todo guardado en la base de datos, yo te voy a retornar una URL que en este caso puede ser http://localhost:8080/medicos (http://localhost:8080/medicos) y el id del médico que ya está guardado.

[01:21] Entonces con un método get tú deberías ser posible de acceder al médico que acabamos de guardar. Entonces esto es lo que tenemos que hacer ahora para digamos cumplir con las buenas prácticas de Rest para el método post. Vamos a comenzar.

[01:41] Voy a borrar esto, y comienzo aquí como siempre, reemplazando void por ResponseEntity. Tengo aquí mi ResponseEntity y lo que tengo que hacer ahora, primero que todo, si yo quiero devolver el cuerpo del médico que he

guardado porque tengo que devolverlo también por estándar, entonces digo aquí Medico medico = médicoRepository con los datos que tiene aquí. Perfecto.

[02:10] Esto me va a devolver un objeto médico de base de datos, de entidad, y ya vimos que yo no quiero obviamente retornar el número de mi entidad de base datos. Yo quiero retornar mi DTO. ¿Qué hago? Le doy a datosRespuestaMedico, igualito, datosRespuestaMedico que sea igual a new datosRespuestaMedico, el médico que acabo de guardar como parámetro.

[02:38] Vemos que aquí hay un error. Y es porque estoy mandando la entidad completa y no los parámetros, entonces aquí voy a copiar el mismo código que ya tengo aquí, no voy a reinventar nada. Entonces vengo aquí y le doy datosRespuestaMedico, le borro un paréntesis que está de más y listo.

[03:03] Ya tengo mi DTO que está siendo creado con el médico que estoy guardando y todo bien. ¿Qué me falta? Me falta retornar: return ResponseEntity. Pero recuerden que no es 200, es 204. Entonces tengo que decirle que retorne un estatus created.

[03:26] Entonces va a decir created, pero si se fijan aquí, created me pide un URI location, o sea, el URL donde esté recurso va a ser encontrado. Entonces yo aquí lo que voy a hacer es crear una nueva URI llamada URL. Y para obtener está URL yo podría hacerlo de dos formas. Yo podría aquí hardcodear como decimos http://localhost:8080/medicos/ (http://localhost:8080/medicos/) y aquí concatenarlo con + medico.getId(); por ejemplo, porque es lo que hay que retornar, getId.

[04:09] Funciona. Podría funcionar, eso lo puedo concatenar y mandarlo a transformar a URI, no hay problema, ¿pero qué pasaría si lo describe un servidor? Entonces ya no va a ser localhost:8080, va a ser el dominio de mi servidor. Para obtener esto Spring también me ayuda con una clase auxiliar, un helper para obtener los datos de mi servidor, y le voy a llamar aquí en los parámetros de este método.

[04:38] Esta clase de aquí es UriComponentsBuilder, voy a llamarlo aquí. Entonces, lo que yo voy a hacer aquí, voy a comentar esto porque esto no va a funcionar, voy a repetir este código de aquí, mi URI, y voy a decir uriComponentsBuilder.path, porque yo necesito mi path aquí, y va a ser "medicos/".

[05:10] Y aquí viene otra cosa interesante porque yo no le puedo dar digamos un id fijo porque si no, todos los URL creados apuntarían a ese mismo id. Y esto tiene que ser dinámico entonces lo que hago es poner entre llaves y con el parámetro dinámico id, aquí termina mi path. Aquí están bien lo que yo le voy a poner aquí es buildAndExpand.

[05:38] Y aquí le voy a decir medicos.getId y finalmente toUri. De esta forma yo ya tengo mi URL creada y eso es lo que yo se lo voy a mandar aquí a created. Entonces voy a borrar esto porque que no me va a servir de nada mi código. Aquí yo estoy creando la URL dinámicamente, el recurso siempre va a ser el mismo, ese mismo recurso que tengo mapeado aquí, médicos, médicos, el id es dinámico porque se lo estoy mandando aquí con este método, esta clase.

[06:16] ¿Qué más me pide? Me está pidiendo un body aquí abajo, y el body obviamente va a ser mi datosRespuestaMedico. Y finalmente eso sería todo lo que tendría que hacer para devolver los datos de un médico que acabo de crear. Ahora, por ejemplo, vemos que aquí me está dando un warning en este ResponseEntity. ¿Esto por qué es?

[06:38] Porque ResponseEntity, si entramos a verlo, acepta un parámetro genérico. Eso quiere decir que yo puedo hacer mi código mucho más estandarizado u organizado, si yo le digo que este ResponseEntity lo que va a responder es un objeto del tipo datosRespuestaMedico.

[07:04] Con esto, yo le digo esto ResponseEntity solamente va a responder este tipo de DTO. Si yo le cambiara esto, por ejemplo por datosRegistroMedico,

entonces me va a dar un error de compilación. ¿Por qué? Porque no es el DTO que espera.

[07:18] Entonces por buena práctica es bueno también especificar a nivel de ResponseEntity en el código qué tipo de objeto es el que yo espero retornar a mi cliente. También se vuelve un poco más seguro porque ya no hay tantos errores a nivel de prometemos estás hardcodeando y te equivocas de DTO por ejemplo.

[07:38] Bueno, vamos a guardar, ya no vamos a perder más tiempo hablando aquí. Tenemos la creación de DTO, está guardando. Si se dan cuenta no hay nada, no hay ciencia nueva aquí, es lo mismo que me visto, pero agregándole un poco más de detalles, que van a hacer este código mucho más entendible, mucho más fácil de leer porque está adaptado a los estándares de Rest y HTTP, venimos aquí y ahora voy a guardar un nuevo médico.

[08:06] A este lo voy a llamar “Alexis Sandoval”. Reemplazo aquí, Alexis Sandoval, y el número de documento que no puede ser el mismo, número de teléfono también lo vamos a cambiar y listo. Vamos a ver qué nos dice ahora. Vemos que nos dio un bad request. ¿Por qué?

[08:30] Vamos a ver qué es lo que nos dice. La validación falló para el argumento de aquí porque claro, el documento es mucho más grande que el que estamos aceptando en nuestra validación. Correcto. Entonces vamos a enviar otra vez y vemos aquí que recibimos el 201 created.

[08:46] Y vemos aquí todos los parámetros que hemos recibido, incluso el nuevo id de la base de datos. ¿Por qué? Porque este id fue auto generado por la base de datos y ya nos lo ha devuelto aquí pero la magia también está en otro lado. Por este lado tenemos los headers y aquí en los headers, tenemos algo muy interesante que quiero que vean y es esto de aquí.

[09:11] Esto de aquí es el header que les comenté donde nosotros podemos encontrar al médico que acabamos de guardar ahora, por ejemplo, yo voy

ahora a crear. Voy a regresar aquí y voy a crear un nuevo request. Este va a ser un get y va a ser http://localhost:8080/medicos/10 (http://localhost:8080/medicos/10).

[09:33] ¿Por qué? Porque ese es el que yo acabo de guardar. Si le doy un enviar, me va a decir método no permitido, method not allowed y otro estado de error: 405. ¿Por qué es eso? Vamos a descubrirlo en el siguiente video.